

## Library Designs for Generic C++ Sparse Matrix Computations in Iterative Methods

*Roldan Pozo*

<http://math.nist.gov/pozo>

Applied and Computational Mathematics Division  
National Institute of Standards and Technology

## Linear Algebra C++ Software

- **LAPACK++** [Dongarra, Pozo, Walker]
  - linear systems, eigenvalues, least squares
- **IML++** [Dongarra, Lumsdaine, Pozo, Remington]
  - Krylov solvers: **PCG**, **GMRES**, **QMR**, etc..
- **SparseLib++** [Pozo, Remington, Lumsdaine]
  - various sparse matrix representations
  - preconditioners (**ILU**, **IC**, **Jacobi**)
- **MV++** [Pozo]
  - micro-kernel matrix/vector classes
- **TNT: Template Numerical Toolkit for Linear Algebra** [Pozo]

## Software Design Goals

- separate the data implementation details out of the fundamental algorithms
- describe library algorithms in terms of generic objects (inheritance or C++ templates)
- specify only the interface
- utilize compile-time binding (templates) or run-time binding (virtual functions) to determine implementation details

Need good design to choose proper objects and abstractions to balance

- performance
- reusability
- generality
- control
- flexibility

## Example CG from IML++

```
template < class Matrix, class Vector, class Preconditioner, class Real >
int CG(const Matrix &A, Vector &x, const Vector &b, const Preconditioner &M,
int &max_iter, Real &tol)
{
  // ...

  for (int i = 1; i <= max_iter; i++) {
    z = M.solve(x);
    rho(0) = dot(x, z);

    if (i == 1)
      p = z;
    else {
      beta(0) = rho(0) / rho_1(0);
      p = z + beta(0) * p;
    }

    q = A*p;
    alpha(0) = rho(0) / dot(p, q);

    x += alpha(0) * p;
    r -= alpha(0) * q;

    if ((resid = norm(r) / normb) <= tol) {
      tol = resid;
      max_iter = i;
      return 0;
    }
  }

  // ...
}
```

## Vector, Matrix interface requirements

$Vector \leftarrow Matrix \operatorname{operator} * Vector$

$Vector \leftarrow Matrix^T \operatorname{operator} * Vector$

$Vector \leftarrow Scalar * Vector$

$Vector \leftarrow Vector +/- Vector$

$scalar \leftarrow Vector::\operatorname{norm}()$

$Vector \leftarrow \operatorname{operator}=(Vector)$

$scalar \leftarrow \operatorname{dot}(Vector, Vector)$

$Vector \leftarrow Preconditioner::\operatorname{solve}(Vector)$

$Vector \leftarrow Preconditioner::\operatorname{trans\_solve}(Vector)$

## Level 3 Sparse Blas[1,2] Toolkit

- sparse matrix products,

$$C \leftarrow \alpha \operatorname{op}(A) B + \beta C$$

- solution of triangular systems,

$$C \leftarrow \alpha D \operatorname{op}(A)^{-1} B + \beta C$$

$$C \leftarrow \alpha \operatorname{op}(A)^{-1} DB + \beta C$$

- reordering of a sparse matrix (permutations),

$$A \leftarrow A \operatorname{op}(P)$$

- conversion of one data format to another,

$$A' \leftarrow A$$

where  $\alpha$  and  $\beta$  are scalars  
 $B$  and  $C$  are dense (multiple vectors)  
 $D$  is a (block) diagonal matrix,  
 $A$  and  $A'$  are sparse  
 $\operatorname{op}(A)$  is either  $A$  or  $A^T$ .

[1] S. Carney, M. Heroux, G. Li, R. Pozo, K. Remington, K. Wu  
 [2] I. Duff, M. Marrone, G. Radicati

## IML++ Performance[1]: GMRES with diagonal preconditioning on SEIMIC (N=3,1752 nz=190,258)

SPARC-5:

	Time	TM in (sp) MV	TM in BLAS1
C++	439	61.0	353.4
C++, FT sparse BLAS	467	99.1	340.7
C++, OPT BLAS1	254	61.4	165.3
C++, FT sp BLAS and OPT BLAS1	286	99.3	159.8
Fortran, OPT BLAS1	261	85.2	158.3
Fortran -fast, OPT BLAS1	208	31.6	160.1

[1]Guangye Li, Cray Research, Inc.

## IML++ Performance: GMRES with diagonal preconditioning on SEIMIC (N=3,1752 nz=190,258)

Cray T3D

	Total time	TM in (sp) MV	TM in BLAS1
C++	163	26.3	129.3
C++ with OPT BLAS1	64.6	26.4	32.5
Fortran, OPT BLAS1	60	23.5	32.5

[1]Guangye Li, Cray Research, Inc.

## Template Numerical Toolkit (TNT) for Linear Algebra

- successor to Lapack++, SparseLib++, MV++, and IML++
- supports templated vectors and matrices
- builds upon the Standard Template Library (STL) of ANSI C++
- integrated package for dense and sparse matrix computations
- multi-level interface to various level BLAS kernels
- larger collection of matrix and vector classes
- common high-level interface, e.g.  $A(i, j)$ , for sparse and dense structures

## TNT Iterative Methods Module (IMM)

- new *expert* interfaces utilize split-phase computation (separate initialization and iteration) for greater efficiency and flexibility
- user-specific functions for  $A * x$ , e.g.  $f(x, t_1, t_2, \dots)$ , can be still be used in type-safe way, *without unnecessary copying or global variables*
- visibility of  $t_1, t_2, \dots$  is application-controlled by the glue interface between IMM routines and  $f()$
- conversion tests are performed outside iteration and and orchestrated by the calling application
- uses conventions from the ANSI C++ standard and the Standard Template Library (STL)
- generic preconditioner  $M$  is now a function object

## TNT code examples

### • Simple driver

```
int CG(A, x0, b, maxit, tol, [M]);
```

- $A$  is a operator for  $A * x$
- $x_0$  is initial and final  $x$
- $M$  is the preconditioner (optonal)
- return value is the number of iterations performed

### • Expert driver

```
CG_method P(A, x0, b, [M]);
```

```
for (i=0; i<maxit; i++)
{
    ...
    P.iterate();
    ...
    // general convergence test with
    // P.x() or P.r()

    if (...) break;
}
```

## Example IMM Constructors

```
BiCG_method(A, x0, b, At, M, Mt, r0);
```

```
QMR_method(A, x0, B, At, M1, M1t, M2, M2t, w)
```

```
CGS(A, x0, B, M, r0)
```

- generic matrices and preconditioners
- $A$  needs only  $A * x$ ; preconditioner  $M$  needs only operator() defined, i.e.  $x=M(y)$  means “solve  $Mx = y$ ”

```
CS_method U(A, x0, b, M);
CGS_method V(A, x0, b, M);
```

```
while(1)
{
    ...
    U.iterate();
    V.iterate();

    // generat check with r, x, or
    // number of iterations from U or V
    if (...) break
}
```

## Integrating with user-supplied matvecs

```
void my_Ax(double *x, double *y, int N, T1 t1, T2 t2, T3 t3);
void my_M(double *x, double *y, int N, T4 t4, T5 t5, T6 t6);
void my_Mt(double *x, double *y, int N, T7 t7);
```

### Specify interface

```
class my_Ax_
{
public:
    T1 &t1_;
    T2 &t2_;
    T3 &t3_;

    my_Ax_(T1 &t1, T2 &t2, T3 &t3): t1_(t1), t2_(t2), t3_(t3) {}

public:
    Vector operator*(const Vector &x)
    {
        Vector y(x.size());
        my_Ax(x.begin(), y.begin(), x.size(), t1_, t2_, t3_);
        return y;
    }
}
```

## Application Example

```
void my_Ax(const double *x, double *y, int N,
           const Grid &G, R_level R);

void my_M(const double *x, double *y,
           const ILU_factor &F, double error &e);
```

```
class my_Ax_
{
public:           // control visibility
    const Grid &t1_;
    R_level &R_;

public:
    my_Ax_(const Grid &G, R_level R_): G_(G),
        R_level_(R_level) {}
    Vector operator*(const Vector &x)
    {
        Vector y(x.size());
        my_Ax(x.begin(), y.begin(), G_, R_);
        return y;
    }
}
```

## Application Example (cont'd)

```
main()
{
    // ...
    my_Ax_ A(G, R);
    my_M_ M(t, v);

    CG_method V(A, x0, b, M);           // expert driver

    for (int i=0; i<max_iter; i++)
    {
        // ...
        V.iterate();

        // if not converging significantly, increase the
        // refinement level...

        if ( i > 30 && norm2(V.r()) > threshold)
            A.R +=1;

        // generic test for convergence
        // if (...) break
    }
}
```

## IMM CG code

```
template <class Matrix, class Vector, class Precond>
class CG_method
{
private:
    int num_iter;
    Vector p, z, q;
    Vector r;
    Vector::value_type alpha, beta, rho, rho1;

public:
    CG_method(const Matrix &A, const Vector &x0, const Vector &b,
              const Precond &M);

    void iterate();

    int num_iterations();
    const Vector &x() const;
    const Vector &r() const;
};

template <class Matrix, class Vector, class Precond>
CG_method<Matrix, Vector, Precond>::CG_method(const Matrix &A,
                                              const Vector x0, const Vector &b, const Precond &M)
{
    x = x0;
    r = A*x - b;
    alpha = beta = rho = rho1 = 0;
    num_iters = 0;
}
```

### IMM CG code (cont'd)

```

template <class Matrix, class Vector, class Precond>
void CG_method<Matrix, Vector, Precond>::iterate()
{
    z = M(r);

    if (num_iters == 0) p=z;
    else
    {
        beta = rho / dot_product(p, q);
        p = beta * p + z;
    }

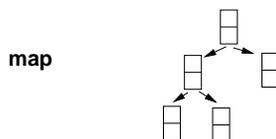
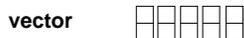
    q = A*p;
    alpha = rho / dot_product(p,q);

    x += alpha * p;
    r -= alpha * q;

    rho1 = rho;
}
    
```

### Standard Template Library (STL)

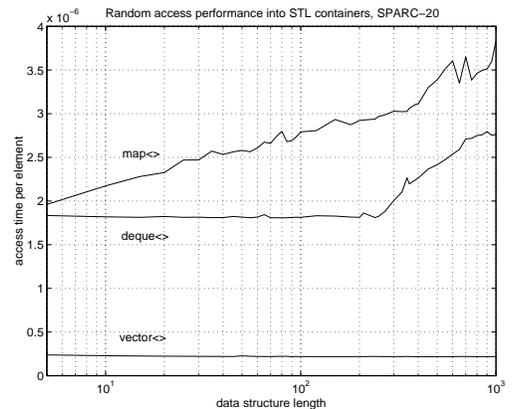
- based on C++ template mechanism (no run-time overhead)
- component of the ANSI C/C++ standard
- collection of generic (type-independent) algorithms and data structures (e.g. linked lists, queues, etc.)
- single algorithm (e.g. find()) works for most STL data structures
- for M data structures and N algorithms, there are M+N codes, *not* M\*N.
- For dense and sparse linear algebra, most useful STL components are vector<>, list<>, and map<> containers.



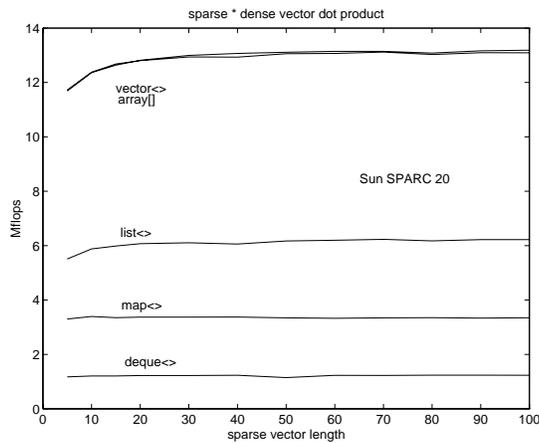
### TNT Matrix Representations

- **Dense**
  - Vector
    - \* arbitrary base
    - \* optional bounds check
    - \* can reference subsections
    - \* fast index access
  - Col\_Matrix: contiguous column storage, (Fortran)
  - Row\_Matrix: contiguous row storage
  - C\_Matrix: array of non-contiguous vectors
  - Index : index range (unit stride)
  - Region: submatrix, subvector references (block algorithms)
- **Sparse**
  - sparse\_Fortran\_vector
  - sparse\_STLmap\_vector
  - sparse\_STLlist\_vector
  - sparse\_STLvector\_vector
  - Coordinate\_Matrix (map, vector, list, Fortran)
  - CompressedRow\_Matrix “ ”
  - CompressedCol\_Matrix “ ”

### STL Performance



## TNT sparse/dense vector dot-product performance



	iteration cost	random cost	insert cost
map	8x	$O(\log N)$	$O(1)$
vector	1x	$O(1)$	$O(N)$
list	5x	$O(N)$	$O(1)$

## Object-based Sparse BLAS

### FORTRAN

```

DO 100 I=1,NZ
100  SUM = SUM + VAL(I) * Y(INDX(I))

```

### C++

```

for (iterator p = x.begin(); x != x.end(); p++)
    sum += (*p).val * y( (*p).indx );

```

C++ code works for generic sparse vectors (e.g. based on STL linked-list, vector, map).

## C++ performance Issues

- handling of temporaries ( $y \leftarrow A * x - b$ )
  - lazy evaluation
  - proxy classes
  - template optimizations
- limited operator overloading
  - binary operators (no  $y += A * x$ )
  - no destination specification (e.g.  $A = x * y$ )
  - use +=, \*=, rather than + and \*
- return value optimizations
  - implemented by some compilers
  - still not useful for vectors or matrices
- deep copying vs. shallow copying
  - shallow copying (share semantics)  $A=B$  refer to the same data
  - deep copying most consistent with C++ model, but can be expensive if not implemented right

## C++ Template Caveats

- not universal support from compilers (yet...)
- increased compile time
- cannot overload on ambiguous types
- code bloat: large executables
- must provide source code
- template parameters must match exactly to trigger

## Trade-offs between C++ run-time and compile-time polymorphism

- Inheritance
  - classic OO design
  - provides greater run-time flexibility via late-binding
  - more challenging to integrate with external packages
  - some run-time overhead
- Templates
  - more recent addition to C++
  - independent of OO design
  - all *type* information must be known at compile time
  - can be efficiently compiled
  - no run-time overhead

## Summary

- TNT is a new design to for generic linear algebra codes
- it utilizes the template mechanism of C++, together with components of the Standard Template Library (STL) of ANSI C++
- the Iterative Methods Module (IMM) utilizes a split-phase technique for generic algorithms to increase flexibility and control; the integration with existing applications is accomplished via class-specification interfaces
- the Sparse Matrix Module (SMM) utilizes various implementations of matrix and vector objects, allowing greatest the greatest freedom in choosing algorithm/data-structure combinations
- the template mechanisms still allows for specialization on common data types (double, complex) to utilize optimized BLAS kernels whenever possible

## Conclusion

- our goal is to develop a framework in which algorithms are described at a higher-level abstraction
- the resulting libraries are more flexible, easier to understand, and utilize in application codes
- with careful design (and utilizing the latest compiler technology, templates mechanisms, and object optimizations) one can implement such generic libraries with competitive performance.